SKA SDC2 reproducibility awards

In partnership with the Software Sustainability Institute (SSI), we will be awarding a set of **reproducibility awards** to all teams whose pipelines demonstrate best practice in the provision of reproducible results and reusable methods. An essential part of the scientific method, reproducibility leads to better, more efficient science. Reusability generalises this principle to create software that can be adapted by others, allowing previous work to be built upon for the future: a key feature of open science.

Reproducibility awards will run in parallel and independently from the SDC2 score, and there is no cap on the number of teams to whom the awards can be given. All pipelines will be evaluated at the close of the challenge using the criteria set out below. These criteria can also be used for self-assessment by teams during the challenge. Colour coding is used to indicate three levels of award: bronze, silver and gold.

All parts of the software pipeline that have been developed by each team will be evaluated. This includes packages that the team have written and code that interacts with third party packages, but does not include any third party packages themselves.

We encourage each team to discuss early on in the challenge the overall **architecture and design** of their software pipeline, in order to identify and agree upon which practices will be put in place during pipeline development. The SSI provide a fantastic collection of guides to software best practice; the top five don'ts of software development is a great place to start. Several more guides are linked below alongside the relevant award criteria.

Table key:

🟧 Bronze level

⬜ Silver level

🟨 Gold level

	Reproducibility of the solution Can the software pipeline be re-run easily to produce the same results? Is it: ● Well-documented *Research software documentation best practice* ● Easy to install *Top tips for packaging software* ● Easy to use *Top tips for documentation*	
Well-documented	High-level description of what/who the software is for is available	🟧
	High-level description of what the software does is available	🟧
	High-level description of how the software works is available	🟧
	Documentation consists of clear, step-by-step instructions	🟧
	Documentation gives examples of what the user can see at each step e.g. screenshots or command-line excerpt	⬜
	Documentation uses `monospace` fonts for command-line inputs and outputs, source code fragments, function names, class names etc	🟨
	Documentation is held under version control alongside the code	🟨
Easy to install	Full instructions provided for building and installing any software	🟧
	All dependencies are listed, along with web addresses, suitable versions, licences and whether they are mandatory or optional	🟧
	All dependencies are available	🟧
	Tests are provided to verify that the installation has succeeded	⬜
	A containerised package is available, containing the code together with all of the related configuration files, libraries, and dependencies required. *Using .e.g. Docker/Singularity*	🟨
Easy to use	A getting started guide is provided outlining a basic example of using the software *e.g. a README file*	🟧
	Instructions are provided for many basic use cases	⬜
	Reference guides are provided for all command-line, GUI and configuration options	🟨

Reusability of the pipeline

Can the code be reused easily by other people to develop new projects? Does it:

- Have an open licence *Choosing an open source licence*
- Have easily accessible source code *Choosing a repository for your project*
- Adhere to coding standards *Writing readable source code*
- Utilise tests *Testing your software*

Open licence	Software has an open source licence *e.g. GNU General Public License (GPL), BSD 3-Clause*	🟧
	Licence is stated in source code repository	⬜
	Each source code file has a licence header	🟨
Accessible code	Access to source code repository is available online	🟧
	Repository is hosted externally in a sustainable third-party repository *e.g. SourceForge, LaunchPad, GitHub: Introduction to GitHub*	⬜
	Documentation is provided for developers	🟨
Code standards	Source code is laid out and indented well	🟧
	Source code is commented	🟧
	There is no commented out code	🟧
	Source code is structured into modules or packages	⬜
	Source code uses sensible class, package and variable names	⬜
	Source code structure relates clearly to the architecture or design	🟨
Testing	Source code has unit tests	🟨
	Software recommends tools to check conformance to coding standards *e.g. A ‘linter’ such as PyLint for Python*	🟨